DOI:
10.1109/MCSE.2008.82

Document Version
Peer reviewed version

Provenance: The Bridge Between Experiments and Data

Simon Miles[1], Paul Groth[2], Ewa Deelman[2],
Karan Vahi[2], Gaurang Mehta[2], Luc Moreau[3]
[1] *Department of Computer Science, King's College London, UK*
[2] *Information Sciences Institute, University of Southern California, US*
[3] *Electronics and Computer Science, University of Southampton, UK*

Current scientific applications are often structured as workflows and rely on workflow systems to compile abstract experiment designs into enactable workflows that utilise the best available resources. The automation of this step and of the workflow enactment, hides the details of how results have been produced. Knowing how compilation and enactment occurred allows results to be reconnected with the experiment design. We investigate how provenance helps scientists to connect their results with the actual execution that took place, their original experiment and its inputs and parameters.

Introduction

Today, more than ever, managing data is an increasingly difficult both in the scientific and business domains, and even in our personal lives. Projects such as LIGO (the Laser Interferometer Gravitational Wave Observatory) [2, 5], ESG (Earth Systems Grid) [3] and many others are collecting petabyte-scale data sets. Data interpretation issues are not only geared towards raw data description but also towards the description of the derived data products and of the processes that created them. An additional complexity is introduced by the execution systems, which can be very heterogeneous and distributed across many locations. In order to be able to understand the data, scientists need to be able to interpret *metadata* about the data as well as its *provenance* (how the data came to be as it is).

As an illustrative example of importance of provenance of data, consider digital photographs, often used as evidence in courts. Photo metadata is regularly embedded in files, such as date, position, or camera settings. It helps support the judicial case since it provides circumstantial evidence about the subject of the photo. However, recent court cases have highlighted that image processing had altered the fundamental nature of photographs, by erasing objects from them. To be able to ascertain the authenticity and validity of such evidence, it is necessary to understand the different derivations that have been applied to images, and in what way they may impact their quality as evidence material: adjusting brightness may be acceptable, while cropping may not be. The same principle applies to scientific data, where the validity of results depends on the details of the experiment conducted.

Workflow Technologies and Applications

In science, the trend is towards the use of workflow technologies to manage the data processing. Scientific workflows enable the description and management of complex analyses. They describe the computations, their parameters, input and output data, and data or control dependencies between them. Workflows are managed by software systems

that follow the workflow dependencies and execute the computations on the desired data. Workflow technologies are not only useful for the automation of complex scientific analyses but they also provide us with the opportunity to capture the transformations performed on the data.

To illustrate the complexity of today's analyses, we examine a popular astronomy application, Montage [4]. Montage produces science-grade mosaics of the sky on demand. This application can be structured as a workflow that takes a number of images, projects them, adjusts their backgrounds, and adds the images together. A mosaic of 6 degrees square would involve processing 1,444 input images, require 8,586 computational steps and generate 22,850 intermediate data products. In order to verify the quality of the final mosaic, a scientist may need to check that a particular input image was retrieved from a specific archive, that the parameters for the reprojections were set correctly, that the execution platforms used did not include processors with a known floating point processing error, etc.

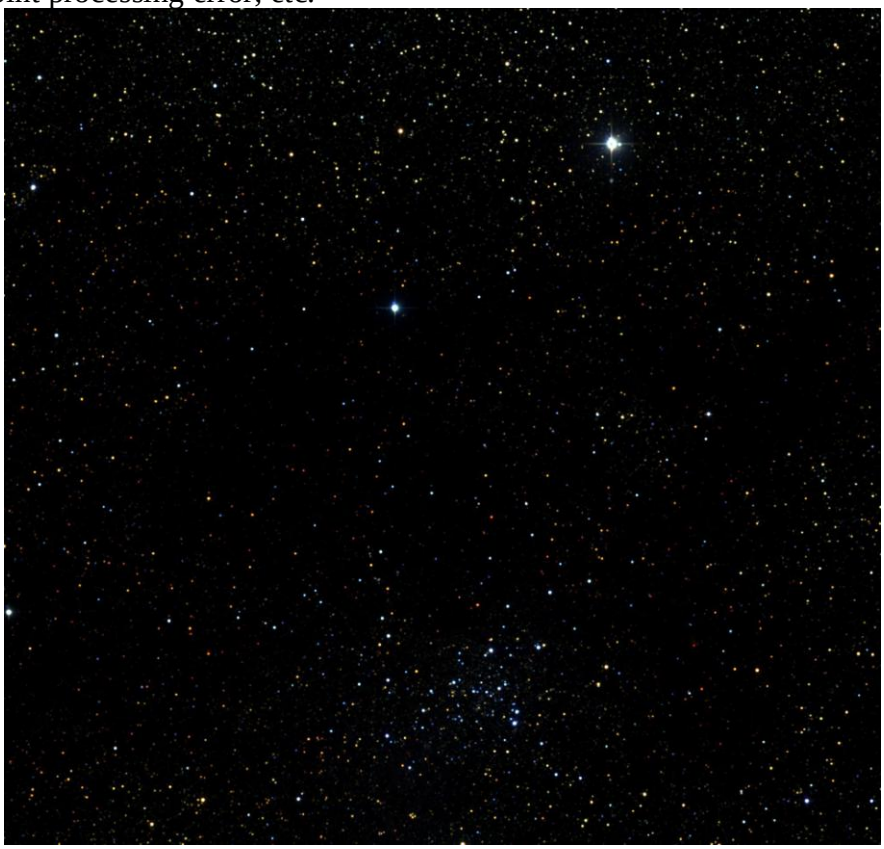

Figure 1: Image of the Sky Near Galaxy M16 Produced by Montage

Workflow Abstractions and Compilers

Given the complexity of workflows with thousands of computational steps, possibly executing across multiple, dynamically changing distributed resources, it is infeasible for users to directly define the executable workflow. The resources are often shared with other users and may become suddenly unavailable due to network failures or policy changes. Thus, researchers often use "workflow compilers" such as Pegasus [6, 7] to

generate the executable workflow based on a high-level, resource-independent description of the end-to-end computation (an *abstract workflow)*. This approach gives scientists a computation description that is portable across execution platforms and can be mapped to any number of resources. However, the additional workflow mapping also increases the gap between what the user defines and what is actually executed by the system and thus complicates the interpretation of the results: the connection between the scientific results and the original experiment is lost.

A preliminary paper describing our approach was published at [10] and included a prototype-based performance evaluation. This paper focuses on the broad principles of the approach and its benefits for scientist users. We present a solution that not only describes how the computations were executed but also how these computations were generated and how they relate to the description provided by the scientist.

Workflow Compilation/Refinement

As the basis of this work, we use the Pegasus workflow compiler system (Planning for Execution in Grids, 2007), which maps high-level, abstract workflow descriptions onto the available distributed resources. The abstract workflow provided by the user, portal, or another workflow composition system [8] is resource independent. The abstract workflow specifies the computations, their input and output data, and interdependencies between them without indicating where the computations take place, or where the data is located. A very simple workflow description could define computing the function *F* on an input *x*, generating the output *Y* and *storing it on the storage system resource S* (see Figure 2 (a)). In Figure 2, ovals denote workflow components, document icons denote data and silos denote resources. The basic process of generating an executable workflow involves the following steps:

a) Find *x* which can be located at zero or more storage system resources: {S_1, S_2, …}
b) Find where F can be computed given that the computing site resources are: {C_1, C_2, …}
c) Choose a computation site *c* and a storage system *s* subject to constraints (performance, space availability etc.)

As a result, the following executable workflow will be constructed.

1. Copy *x* from *s* to *c*
2. Move *F* to *c*
3. Compute *F(x)* at *c*, obtaining *Y* at *c*.
4. Move *Y* from *c* to *S*
5. Register *Y* in data registry

Once the workflow is generated, the descriptions of steps a-c gives us the provenance of the executable workflow whereas once the workflow is executed the descriptions and execution details of steps 1-5 give us the provenance of the workflow's output, Y.

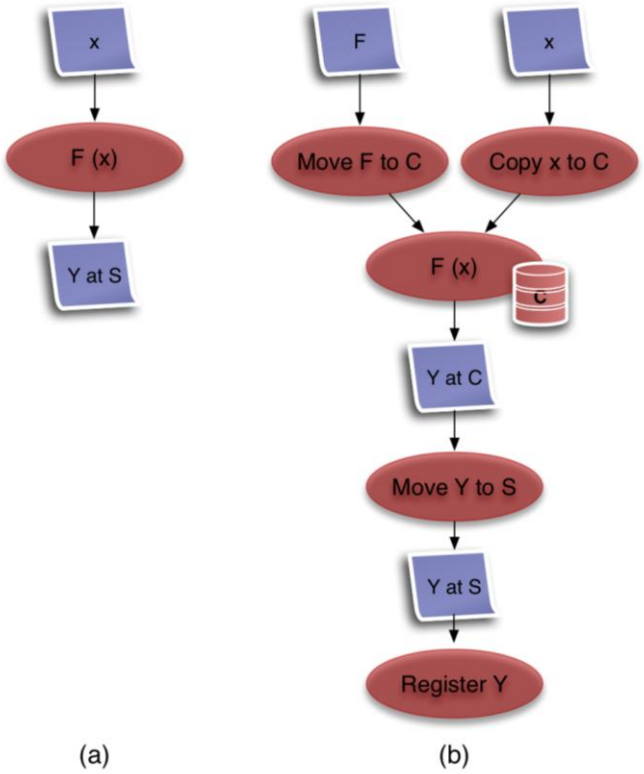

Figure 2: An abstract workflow (a) and its concrete version (b)

Understanding the provenance of *Y* requires understanding its connection to the original workflow, especially where problems occur. Even with this simple workflow, things can go wrong: *x* was not found at *s*, *F(x)* failed, *c* crashed, or there was enough space at *S*. Given these 4 error messages, the user may only understand the second and last messages since they relate to the original request. However, the fact that *x* was not at *s* is harder to interpret, especially if there is another copy of *x* somewhere else. The "*c* crashed" message can be particularly hard to interpret. Obviously, the workflow mapping and execution system can try to shield the user from some of these failures and try to recover, however, at some point this may not be possible, which is when understanding the provenance of a data item becomes important.

Pegasus and workflow refinement

Above, we provided an example of the mapping from an abstract to an executable workflow. In this section, we present the steps that Pegasus goes though to compile an abstract workflow description into an executable workflow. This process is also known as the *refinement* process, as the information in the abstract workflow is refined to the point of execution. To illustrate the changes made to the workflow as part of the refinement process, we use part of the Montage workflow shown in Figure 3, which reprojects images and takes their differences. We illustrate the various changes to the workflow during refinement in Figure 4. Although, we use Pegasus as an illustrative example of workflow compilation, other workflow systems, such as for example Askalon [13] perform workflow restructuring.

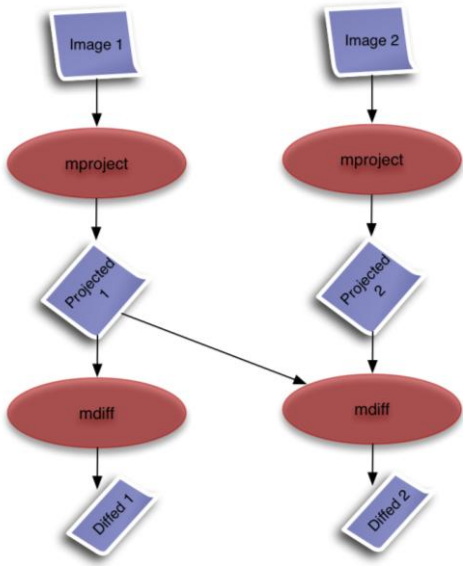

Figure 3: An Example Montage Workflow

The main refinement steps in Pegasus are:

- **Reduction (Figure 4a):** eliminates processing steps when intermediate data products have already been generated (by another workflow or previous execution of this workflow) and can be reused. In the example, files "Projected 1" and "Diffed 1" are found to already exist on a storage system, so they do not need to be recomputed.
- **Site Selection (Figure 4b):** chooses the computational resources on which to execute the jobs described in the workflow. This involves finding available resources and determining where the required executables are already installed or can be staged in. The workflow nodes are annotated with their target execution sites. In our example, resources including R1 and R2 are available, so Pegasus uses these.
- **Data Staging (Figure 4c):** after consulting a data registry, this step selects sources of input data for computations, and adds nodes to the workflow to stage this data in and out of the computation sites. Nodes are also added to transfer output data back to the storage sites. In our example, a node is also added to transfer the intermediate result between execution sites R2 and R1 so that the computation can be invoked at R1. Also the intermediate result "Projected 1" from the first branch of the workflow is staged in for the mdiff computation.
- **Registration (Figure 4c)**: causes final and intermediate data products to be registered in a data registry, by adding registration nodes to the workflow. Data registration nodes are added to the final output data in the workflow. Registration enables workflow-level checkpointing in case of failures as well as helping find the data later.
- **Clustering (Figure 4d)**: In some cases, the granularity of computations (many short run jobs) or the granularity of data staging (many small data transfers) is too fine, causing execution inefficiency. To address this, Pegasus clusters workflow nodes together so that they can be handled as one in execution. In the example, Projected 1 and Projected 2 are at the same site. Thus, the two transfers to resource R1 are clustered together as denoted by the box circling those nodes.

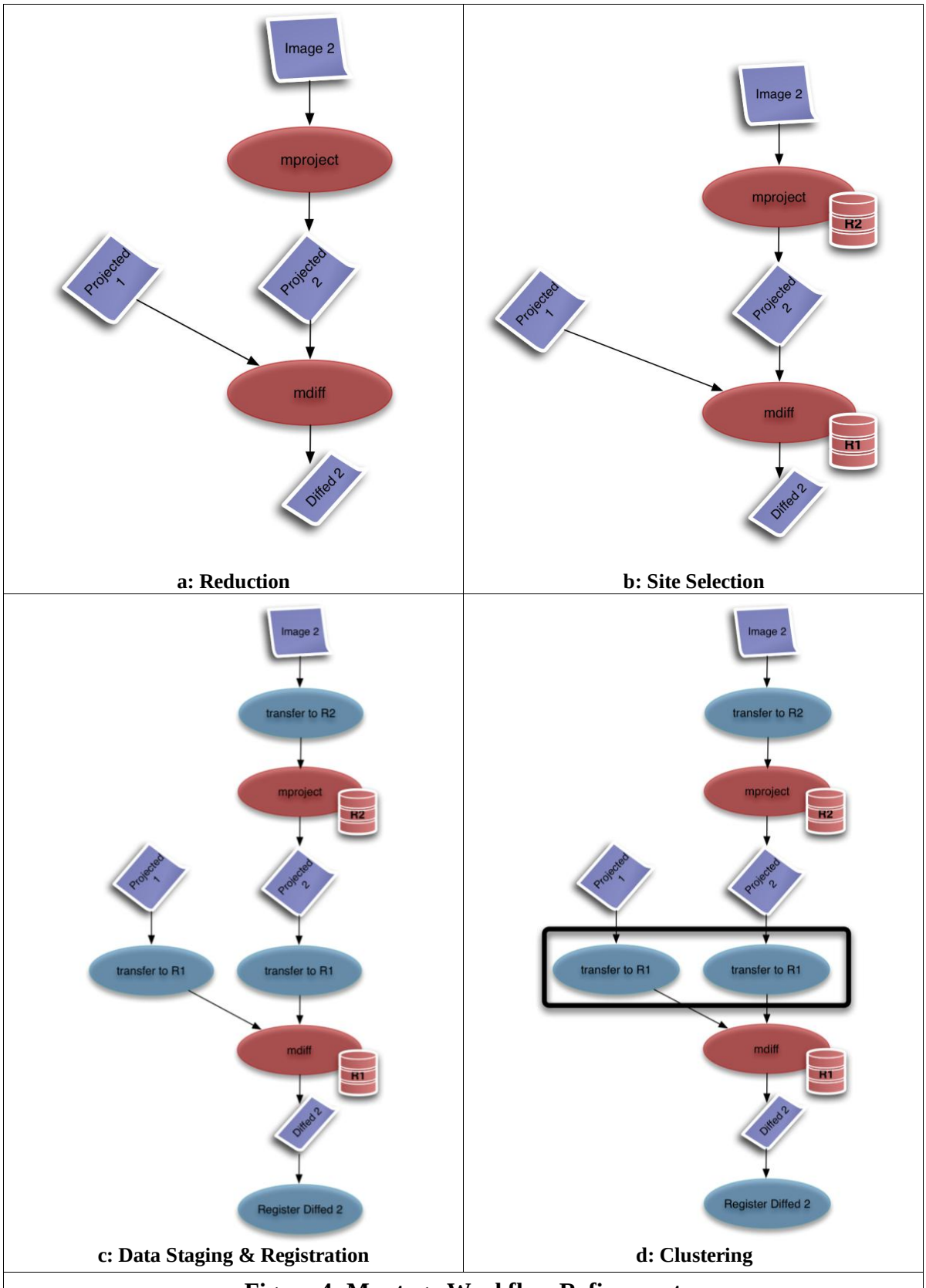

Figure 4: Montage Workflow Refinement

The workflow is now ready for execution and is sent to a workflow engine. In our case, we use Condor DAGMan, which follows the workflow dependencies and executes the directives of the workflow nodes. Once the workflow has executed, a scientist may want to ask several questions related to the provenance of the workflow execution's result. In the next section, we list some of these questions.

Motivating Questions

Based on our collaborations with a variety of domain scientists who use workflows for their analyses, some basic questions are known to be important:

- Which data items were used to generate a particular data product?
- What computations where conducted to generate these data items?
- Where did the computations occur?

There are also questions related to the evolution from the abstract to the executable workflow:

- What happened to this node in my abstract workflow? Why is it not in the executable workflow?
- Which intermediate data product was substituted for the actual computation?
- Why, given that the data was at R2, did the workflow use the data at R1?
- Which abstract node does a particular executable node correspond to?
- Why did the amount of disk at location X diminish so much?
- Why is this intermediate data not in the registry?

The first set of questions can be answered by many provenance systems [11, 12]. However, to answer the latter questions, information must also be recorded about the refinement process, and in a form suitable for answering those questions. We now describe an approach for answering both the first and second sets of questions.

Documentation of Processes

The provenance of a data item is the process by which that data item came to be as it is. There is a wide range of application areas for which the provenance of data is important [9], and the potential uses cannot all be predicted in advance. We therefore require a generic software system for determining the provenance of data. We developed such a system in the PASOA project [1, 9], and describe it below.

As the resources used to execute a workflow may change between the time of execution and when the provenance is requested, it is often impossible to determine from the resources themselves what has occurred. A crucial part of the system, therefore, is recording documentation about what occurs during execution. We use the term *process documentation* to refer to documentation of the execution of a process (such as a workflow enactment).

The functionality which such a system must support can be broadly split into three. First, as an application executes, it also *creates* a description of its execution, the process documentation. This takes the form of a set of *assertions* about what is occurring. Next, once the documentation has been created, the application records it into tailored persistent storage, called a *provenance store*. Finally, after a data item is produced by an application, users may obtain the provenance of this data item by querying the provenance store. Such a query retrieves the set of assertions that describe the provenance of the data item, i.e. the process by which it was produced. In the next section, we describe how Pegasus has been adapted to include the above sequence of creation, recording and querying.

To support querying, each independent source of process documentation must use the same data model. Workflows can bring together resources from a range of sources and using a range of technologies to tackle a given problem, and we require a data model that is independent of execution technology and application domain.

The provenance store is organized in such a way to enable the provenance of data to be determined from documentation collated from that produced by independent distributed sources. As a whole, the documentation must include a description of the process which occurred. We achieve this independence of sources by viewing applications in terms of the service-oriented architectural (SOA) style adopted by many business and science applications. A service, in technology-independent terms, is a component that takes inputs and produces outputs. In a SOA, *clients* invoke *services*, which may themselves act as clients for other services; the term *actor* is used to denote either a client or a service in a SOA. Actors communicate by exchanging *messages*, and the exchange of one message between actors is an *interaction*. Thus, the execution of an application can be described as the exchange of messages between two actors and transformations that actors perform on the messages they receive in order to generate new messages. Most applications can be viewed as a set of interacting actors, including workflow refinement and enactment, as will be shown for Pegasus below.

The assertions that comprise process documentation can take a limited number of forms. In particular, assertions are created regarding what data has been exchanged between actors in an interaction, and what processing has been done on the inputs to a service to produce the outputs (and, therefore, the relationship between inputs and outputs of a service). They are known respectively as *interaction* and *relationship assertions*.

Taken together, interaction and relationship assertions causally connect the data produced in an application. Thus, unlike other mechanisms for debugging applications, process documentation provides explicit causal connections between data items allowing the process that led to an item being as it is to be traced.

Provenance in Pegasus

Following the PASOA approach, we first considered the Pegasus system in terms of interacting actors. We separately address the *refinement phase*, where an abstract

workflow is refined to an executable one by Pegasus, and the *enactment phase*, where Condor DAGMan enacts the executable workflow.

Refinement Process Documentation

In the refinement phase, Pegasus is modeled as one actor, interacting with each of five *refiners*, also actors, in turn. The model of Pegasus as actor is shown in Figure 5. The interactions are the exchange of partially refined workflows between Pegasus and each refiner, until the output of the final refiner is an executable workflow passed to DAGMan.

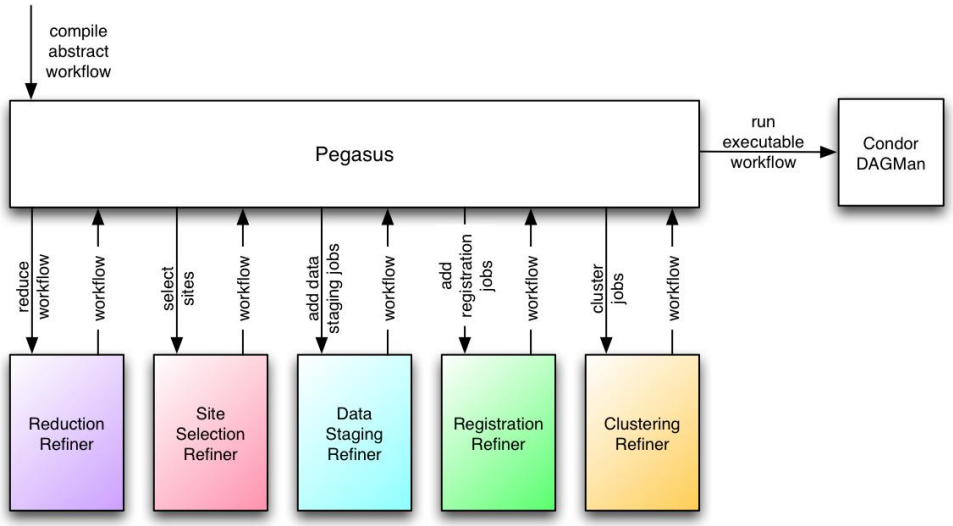

Figure 5: **Pegasus workflow refinement modeled as interacting actors.**

For each refinement step, five recording actions take place. To aid explanation, Figure 6 shows an expanded view of the Site Selection refinement step and Pegasus' invocation of it. Recording actions are labeled A to E, and described below.

- Prior to each refinement, Pegasus records the current, partially refined workflow that is about to be refined further (A).
- It records relationship assertions linking this workflow to the output of the previous refinement (B): these are identical as Pegasus itself does not alter the workflow.
- The refiner records the workflow it receives prior to its refinement (C).
- After refinement, the refiner records the refined workflow (D).
- It also records relationship assertions from each node of the workflow after refinement to the nodes that caused it to be as it is in the pre-refinement workflow (E).

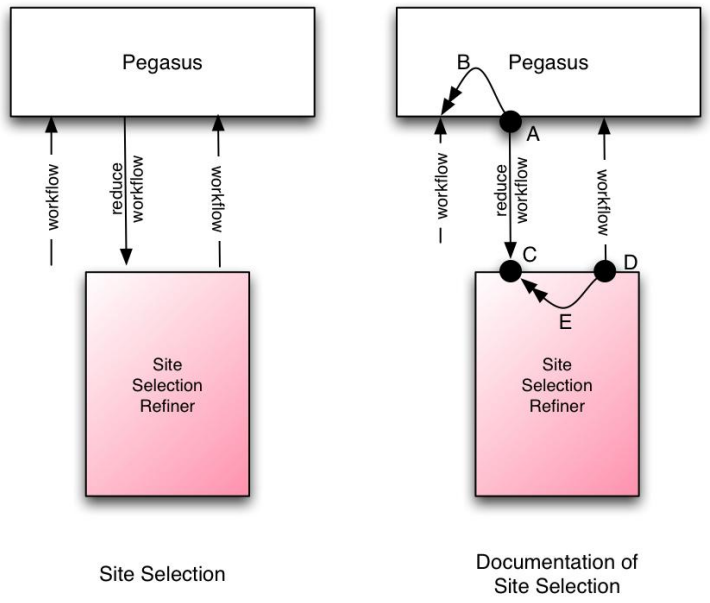

Figure 6: **One refiner's documentation.**

Refinement Relationships

As mentioned in the final step above, each refiner documents the relationships between nodes in the workflow as it was before and after refinement. The type of each relationship gives provenance queriers more information on the refinement that took place.

The following types of relationship are recorded, corresponding to the refinement stages described in Workflow Compilation/Refinement above.

- **identicalTo:** Denotes that a workflow node has not changed in refinement. Its absence for a node in the pre-refinement workflow indicates that the node has changed or been removed in refinement.
- **siteSelectionOf:** Denotes that the post-refinement workflow node is a compute job in the pre-refinement workflow for which the site on which it will run has been chosen and specified.
- **stagingIntroducedFor** Denotes that the post-refinement workflow node is a data staging operation introduced to stage data in/out for the job in the pre-refinement workflow.
- **registrationIntroducedFor:** Denotes that the post-refinement workflow node is a registration operation introduced to follow a stage-out in the pre-refinement workflow.
- **clusteringOf:** Denotes that the post-refinement workflow node is a cluster of jobs combining several jobs in the pre-refinement workflow.

The relationships documented for the example in the Workflow Compilation/Refinement section are summarized in Figure 7. Here, we show the fragment through six states of refinement, from abstract to concrete. The workflow nodes (ovals showing input and

output data) at each stage are related (double-headed arrows) to those in the previous stage, with the relationship type (arrow label) describing the function performed by the refiner that transformed the workflow. By tracing relationships, a querier determines the provenance of each concrete job, and how it relates to the original abstract workflow.

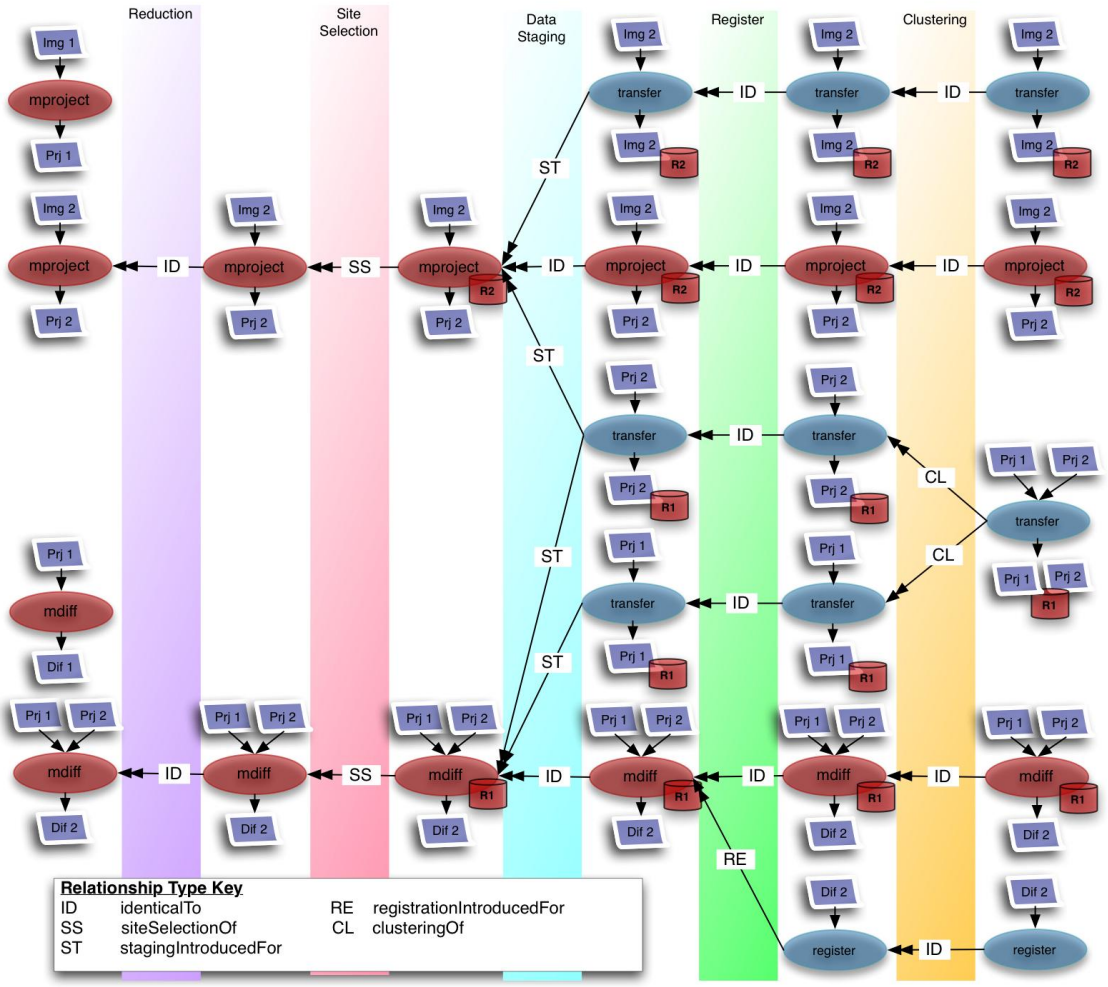

Figure 7: **Relationships recorded during refinement.**

Enactment Process Documentation

The PASOA model for documenting enactment is the same as for documenting refinement. Condor DAGMan is modelled as an actor, interacting with each job in the workflow. DAGMan sends invocation messages, containing command-line arguments including input file names, to the executable jobs, and completion messages are returned from the jobs, containing the names of output files produced.

As with refinement, relationships link together nodes from one step to the next, so that the provenance can later be determined, as in Figure 7. However, here the nodes are the data items processed by the jobs, referred to by filename, rather than the job nodes of the workflow. The relationships between data items depend on the type of job enacted, e.g. a

job which invokes 'gzip' would assert a relationship of type 'gzip' between its output and input.

Refinement and Enactment Connected

The combination of the documentation for workflow refinement and enactment, allows detailed provenance of a data item to be found:

- For each data item, we can find the executable workflow steps that produced it and other data items that contributed to those steps.
- For each workflow step, we can find its connection to the abstract workflow jobs from which it was refined.

We can now revisit our initial questions and find the answers:

- Which data items were used to generate a particular data product?
    - In the case of the example Montage workflow, a scientist could ask what data items were used to generate Diffed 2 and find that Image 2 and Projected 1 were used.
- What computation were conducted to generate these data items?
    - mproject and mdiff were used.
- Where did the computations occur?
    - The mproject computation occurred at the computational resource R2 and the mdiff computation occurred at R1.

All this information can be gleamed from the right-most side of Figure 6 and can be answered by many workflow provenance systems.

There are also questions related to the evolution from the abstract to the executable workflow which can be answered solely by our new comprehensive documentation, which is encapsulated in our example in the left part of Figure 6:

- What happened to this node in my abstract workflow? Why is it not in the executable workflow?
    - In the Montage workflow example, a scientist could specifically ask what happened to the mproject node that takes Image 1 as input. From the documentation, the node was eliminated during workflow reduction.
- Which intermediate data product was substituted for the actual computation?
    - In Figure 6, after the data staging step, a transfer node is added that stages in Projected 1 rather than running mproject on Image 1.
- Which abstract node does a particular executable node correspond to?
    - In Figure 6, for example, one can determine that the mproject node that takes Image 2 as input corresponds to the mproject node running on R2.
- Why is this intermediate data not in the registry?
    - In the above example, one could determine by analyzing the documentation that Projected 2 was not in the registry because Pegasus failed to add a corresponding data registration node.

The following two questions are left for future work. We describe how extensions to our current system could be used to answer them.

- Why did the amount of disk at location X diminish so much?
    - Answering this question requires knowledge of the causality between staging data to a resource and computations generating data and the amount of disk space available at a resource. We envision that models could be built to answer these questions, however, there is an open research problem in as to how a user might formulate such a query.
- Why, given that the data was at R2, the workflow used the data at R1?
    - We intend on extending our system not only to capture the decisions made by the workflow system but also the reason behind those decisions.

We have shown that our system can answer both questions about workflow refinement as well as execution. It is important for a system of this kind that the costs of using it do not outweight the benefits. In recent work, we have analyzed the performance costs of recording process documentation during execution and found the overhead costs sufficiently minor [10].

Conclusions

Understanding the process that ultimately produced a result is critical to correctly interpreting it, and this is particularly important where execution steps are not apparent in the original process design. Provenance is also a key ingredient of scientific reproducibility: colleague can share provenance in order to reproduce and validate each other's results.

In our work, we rely on the proliferation of workflow technologies in the scientific applications and on the workflow systems that automate the process of mapping and executing the applications onto available resources. In this paper, we described our approach to capturing and providing access to the provenance of a workflow: the details chosen for its execution by the Pegasus workflow mapper. We have connected this to the captured documentation of the workflow execution by Condor DAGMan, allowing scientists to determine, for a given data item, by what process it was produced, what abstract workflow led to its execution, and every stage between. This allows the connection between an experiment's results and the original experiment steps to be evident, even when the scientist delegates specifying execution details. We have also integrated the PASOA and Pegasus software implementations.

Future work in the area of provenance needs to address both low and high level issues. The scalability of our approach, while adequate for the problems tackled so far, requires additional techniques for coping with very large data sets, where recording copies of all data passing through the system is infeasible. There also needs to be further effort in easing the process for users in querying for provenance so as to find the information they require, and to include in this not only the processes that have occurred, but the reasons behind their execution. Other issues include the connection between electronic data and

physical components of experiments, each of which have their own, inter-connected, provenance. By keeping the connection between experiment and data, even in complex distributed environments, our work provides a solid basis for these future directions.

[1] "Provenance Aware Service Oriented Architecture," 2006. http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/WebHome

[2] B. C. Barish and R. Weiss, "LIGO and the Detection of Gravitational Waves," *Physics Today*, vol. 52, pp. 44, 1999.

[3] D. Bernholdt, S. Bharathi, D. Brown, K. Chanchio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, and P. Fox, "The Earth system grid: supporting the next generation of climate modeling research," *Proceedings of the IEEE*, vol. 93, pp. 485-495, 2005.

[4] G. B. Berriman, E. Deelman, J. Good, J. Jacob, D. S. Katz, C. Kesselman, A. Laity, T. A. Prince, G. Singh, and M.-H. Su, "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," Proceedings of SPIE Conference 5487: Astronomical Telescopes, 2004.

[5] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis," in *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds.: Springer, 2006.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal*, vol. 13, pp. 219-237, 2005.

[7] E. Deelman, G. Mehta, G. Singh, M.-H. Su, and K. Vahi, "Pegasus: Mapping Large-Scale Workflows to Distributed Resources," in *Workflows in e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds.: Springer, 2006.

[8] Y. Gil, V. Ratnakar, E. Deelman, M. Spraragen, and J. Kim, "Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows," Proceedings of OWL: Experiences and Directions (OWL-ED), Athens, GA, 2006.

[9] S. Miles, P. Groth, M. Branco, and L. Moreau, "The Requirements of Using Provenance in e-Science Experiments," *Journal of Grid Computing*, 2006.

[10] S. Miles, E. Deelman, P. Groth, K. Vahi, G. Mehta, and L. Moreau., "Connecting Scientific Data to Scientific Experiments with Provenance " in *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. Bangalore, India. , 2007.

[11] L. Moreau, B. Ludascher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin Jr, B. Clifford, S. Cohen, and S. Cohen-Boulakia, "The First Provenance Challenge," *Concurrency and Computation: Practice and Experience*, 2007.

[12] Y. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Record*, vol. 34, pp. 31-36, 2005.

[13] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment," *SIGMOD Record*, vol. 34, 2005.